

Algorithmische Graphentheorie

5. Übungsblatt

Jasper Gude

Pia Röttgers

25. Mai 2026

10.5 / 25

Aufgabe 1 – Paarungen (Matchings) in Bäumen

- a) Ein Baum ist ein kreisfreier, zusammenhängender Graph. Ein Baum besitzt ein perfektes Matching genau dann, wenn die Anzahl der Knoten gerade ist und jedes Blatt keine Geschwisterknoten hat.

Die Anzahl der Knoten muss gerade sein, da in einem perfekten Matching jeder Knoten mit einem anderen gematcht wird.

Jedes Blatt muss Einzelblatt sein, da der Elternknoten nur mit einem Blatt gematcht werden kann.

Also ist ein perfektes Matching im Baum eindeutig, da jedes Blatt mit seinem Elternknoten gematcht werden muss. Diese Kante uv ist sicher im Matching enthalten und alle zu u oder v inzidenten Kanten können aus dem Baum gelöscht werden. So entsteht ein neues Blatt, für das die selbe Regel gilt.

- b) Der Algorithmus 1 berechnet ein größtes Matching.

Der Algorithmus ist korrekt, da der Baum T entweder ein perfektes Matching enthält und somit ein perfektes Matching berechnet (Siehe Punkt a)), oder I dont fucking know.

Die Laufzeit für INITIALIZE liegt in $\mathcal{O}(V)$, da sich der Baum in linearer Zeit augmentieren lässt und es $|V| - 1$ viele Kanten gibt.

Die Laufzeit für MATCHATREEREC liegt in $\mathcal{O}(V)$. Der Algorithmus wird genau einmal für jeden Knoten aufgerufen. Wenn eine Kante zum Matching hinzugefügt wird, werden die Knoten und Kanten aus dem Baum

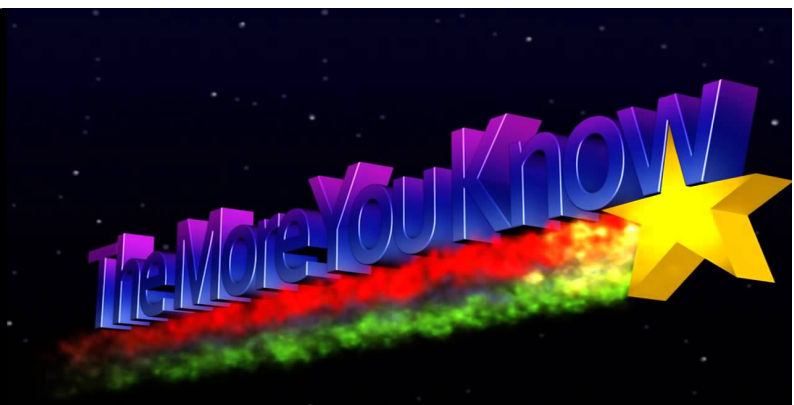
Eltern gibt es in einem Baum nur dann, wenn eindeutig eine Wurzel identifiziert ist. Außerdem schreibt ihr unten, dass dieselbe Regel gilt, aber wenn ihr die (u,v) -inzidenten Knoten löscht, kommt nicht notwendig ein zusammenhängender Graph raus.

Wenn ihr den Baum wurzelt, könnt ihr den Baum in einer Reihenfolge trimmen, die seinen Zusammenhang aufrechterhält, oder ihr könnt zuerst die Eindeutigkeits-Annahme von Bäumen auf Wälder übertragen.

Dafür schöne Beobachtung mit der geraden Knotenzahl =)

2.5/3

...oder weil wir zeigen können, dass bei Knoten wo ein Matching nicht eindeutig ist, alle anderen Matching-Variationen die wir bauen können maximal genauso groß sind.



Algorithmus 1: Größtes Matching in Bäumen

MATCHATEE($T = (V, E)$)

// Den ungerichteten Baum zu einem gerichteten Baum mit parent und
// children pro Knoten und einer Wurzel root augmentieren.

INITIALIZE(T)

return MATCHATEEREC($T, T.root$)

MATCHATEEREC(T, r)

// Rekursiv für die Kinder aufrufen

$m \leftarrow \emptyset$

foreach $v \in r.children$ **do**

$m \leftarrow m \cup \text{MATCHATEEREC}(T, v)$

// Wenn Knoten Blatt ist, dann die Kante zum Elternknoten hinzufügen.

if $r \neq \text{nil}$ **and** $\text{deg}(r) = 1$ **then**

// Die Knoten und inzidente Kanten aus dem Baum entfernen.

$T \leftarrow T \setminus \{r.parent, r.parent.children\}$

$m \leftarrow m \cup \{\{r, r.parent\}\}$

return m

Der Grad eines Knotens ist nicht ohne Aufwand verfügbar. Die meiste Arbeit, um das zu ändern, habt ihr schon gemacht. Ihr könnt im Initialize die Grade (oder alternativ Kinder-Anzahl) aller Knoten in Linearzeit bestimmen und beim Löschen von Knoten die Parent-Pointer verwenden, um diese Information aufrecht zu erhalten.

gelöscht die nicht mehr zum Matching hinzugefügt werden können und somit nicht mehr bearbeitet werden.

Also liegt auch MATCHATEE in $\mathcal{O}(V)$.

4 / 5

- c) Wenn eine Kante rv zu einem Matching hinzugefügt wird, kann keine zu v inzidente Kante hinzugefügt werden. Das müssen wir auch bei unserem dynamischen Programm beachten.

$$\text{OPT}(r) = \max_{v \in r.children} \left\{ rv + \sum_{i \in v.children} \text{OPT}(i) + \sum_{j \in r.children \setminus v} \text{OPT}(j) \right\}$$

Der Algorithmus hält sich an die Eigenschaften eines Matchings, also ist er korrekt.

Der Algorithmus probiert alle Kindknoten von r aus. Geht man mit r im Baum von unten nach oben sind das also höchstens so viele Iterationen wie, Knoten im Baum existieren (ohne die Blätter). Also läuft das Dynamische Programm in $\mathcal{O}(V)$.

Hier kann es passieren, dass sowohl die Kante (r,v) , als auch die Kante (v,i) im Matching landet, also ist v doppelt gepaart. Um das zu vermeiden, müssen wir jeweils zwei OPTs betrachten, sodass wir z.B. den Knoten v von r aus fragen können, was ist dein OPT unter der Annahme, dass rv nicht im Matching ist, und was ist dein OPT, unter der Annahme, dass doch. Die Laufzeitargumentation hält auch mit dieser Modifikation.

3 / 5

Aufgabe 2 – Hamiltonkreise

- a) Ein Kreis hat im allgemeinen mindestens die Länge 3. Da der Turniergraph mindestens 3 Knoten enthält und stark zusammenhängend ist,

liegt jeder Knoten auf einem Kreis mit Länge ≥ 3 .

0.5/ 1

- b) Gegeben ist ein Kreis K mit Länge ≥ 3 . Um K zu verkleinern wählen wir uns drei aufeinanderfolgende Knoten a, b, c aus. Nach Definition gibt es für jedes Knotenpaar eine gerichtete Kante. Wir ersetzen jetzt einen Weg der Länge 2 durch eine Kante die nach Annahme existieren muss. Das machen wir solange, bis der Kreis die Länge 3 hat.

0.5/ 2

c)

/ 1

d)

/ 1

e)

/ 1

f)

/ 1

Aufgabe 3 – Perfekte Matchings in bipartiten Graphen

/ 5